Apple Assembly Line

$1.50

Volume 3 -- Issue 3 December, 1982

In This Issue...

Cross Assemblers continue to appear. We now have ready a version for the Intel 8048, and one for the yet unreleased Rockwell 65C02. More on the latter inside.

Don Lancaster, famous author of many books published by Howard Sams, says the Apple II is probably going to have a greater impact on history than the automobile or television! Perhaps verging on Applolatry, but you will surely enjoy his new book. See Bill's reviews inside.

If I am trying to learn how to program in assembly language, or to increase my skill at it, what (besides AAL) should I read? I strongly recommend Softalk, Nibble, Micro, and Call APPLE. Every month they publish excellent examples of assembly language programs which you can study, modify, and use. As for books, Roger Wagner's, Lance Leventhal's, and Don Lancaster's are my favorites at this time.

Making Internal JMPs and JSRs Relocatable.........Peter Meyer

A machine language routine is said to be relocatable if it can function properly regardless of its absolute location in memory. If a routine contains a JMP or a JSR to an INTERNAL address then it is not relocatable; if it is run in another part of memory then the internal JSR or JMP will still reference the former region of memory. JMPs and JSRs to subroutines at absolute locations (e.g. in the Monitor) do not impair the relocatability of a routine.

I will explain here a technique whereby you can, in effect, perform internal JSRs and JMPs without impairing the relocatability of your routine. There is a small price to pay for this: namely, an increase in the length of your routine. Your routine must be preceded by a 2-part Header Routine which is 43 bytes long. In addition, each internal JSR requires 8 bytes of code, and each internal JMP requires 11 bytes of code.

No tables or other data storage are required, except that three bytes must be reserved for a JMP instruction. These three bytes can be anywhere in memory, but must be at an absolute location. There are three bytes that normally are used only by Applesoft, namely, the ampersand JMP vector at $3F5 to $3F7. Since we are here concerned only with machine language routines in their own right, we can use the locations $3F5 to $3F7 for our own purposes. However, other locations would do just as well.

The technique is fully illustrated in the accompanying assembly language program. This routine consists of three parts:

(1) Header Part 1 (SETUP), which sets up a JMP instruction at VECTOR (at $3F5-$3F7, but could be different, as explained above) to point to Header Part 2.

(2) Header Part 2 (HANDLER), which is a 15-byte section of code whose task is to handle requests to perform internal JSRs and JMPs (more on this below).

(3) The main part of the routine, in which internal JSRs and JMPs (in effect) are performed using macro instructions.

When your routine (including the Header) is executed, the first thing that happens is that Header Part 1 locates itself (using the well-known JSR $FF58 technique), then places a JMP HANDLER at VECTOR. Thereafter a JMP VECTOR is equivalent to JMP HANDLER, and a JSR VECTOR is equivalent to a JSR HANDLER. The HANDLER routine handles requests from your routine for internal JSRs and JMPs. To perform a JSR to an internal subroutine labelled SUBR simply include the following code:

```
HERE   LDA #SUBR-HERE-7 low byte of offset
       LDY /SUBR-HERE-7 high byte of offset
       TSX
       JSR VECTOR
```

```
S-C Macro Assembler (the best there is!)........................$80.00
S-C Macro Cross Assembler Modules
    65C02 Version...............................................$20.00
    6800/6801/6802 Version......................................$32.50
    6809 Version................................................$32.50
    Z-80 Version................................................$32.50
    68000 Version...............................................$50.00
    Requires ownership of S-C Macro Assembler.
    Each disk includes regular and language card versions.
Upgrade from Version 4.0 to MACRO...............................$27.50
Source code of Version 4.0 on disk..............................$95.00
    Fully commented, easy to understand and modify to your own tastes.

Applesoft Source Code on Disk...................................$50.00
    Very heavily commented.  Requires Applesoft and S-C Assembler.

ES-CAPE:  Extended S-C Applesoft Program Editor.................$60.00

AAL Quarterly Disks........................................each $15.00
    Each disk contains all the source code from three issues of "Apple
    Assembly Line", to save you lots of typing and testing time.
    QD#1: Oct-Dec 1980    QD#2: Jan-Mar 1981    QD#3: Apr-Jun 1981
    QD#4: Jul-Sep 1981    QD#5: Oct-Dec 1981    QD#6: Jan-Mar 1982
    QD#7: Apr-Jun 1982    QD#8: Jul-Sep 1982    QD#9: Oct-Dec 1982

Double Precision Floating Point for Applesoft...................$50.00
    Provides 21-digit precision for Applesoft programs.
    Includes sample Applesoft subroutines for standard math functions.

FLASH! Integer BASIC Compiler (Laumer Research)......(regular $79) $49.00
    Special price to AAL readers only, until 12/31/82!
Source Code for FLASH! Runtime Package..........................$39.00

Super Disk Copy III (Sensible Software).............(reg. $30.00)  $27.00
Amper-Magic (Anthro-Digital)........................(reg. $75.00)  $67.50
Quick-Trace (Anthro-Digital)........................(reg. $50.00)  $45.00
Cross-Reference and Dis-Assembler (Rak-Ware)....................$45.00
Apple White Line Trace (Lone Star Industrial Computing)...........$50.00
    (A unique learning tool)

Blank Diskettes (with hub rings)................package of 20 for $50.00
Small 3-ring binder with 10 vinyl disk pages and disks............$36.00
Vinyl disk pages, 6"x8.5", hold one disk each...............10 for $4.50
Reload your own NEC PC-8023 ribbon cartridges..........each ribbon $5.00
Diskette Mailing Protectors........................10-99:  40 cents each
                                              100 or more:  25 cents each

Ashby Shift-Key Mod.............................................$15.00
Paymar Lower-Case Adapter.......................................$37.50
    For Apples before Revision 7 only
Lower-Case Display Encoder ROM..................................$25.00
    Works only Revision level 7 Apples.  Replaces the encoder ROM.

Books, Books, Books........................compare our discount prices!
    "Enhancing Your Apple II, vol. 1", Lancaster.........($15.95)  $15.00
    "Incredible Secret Money Machine", Lancaster..........($7.95)   $7.50
    "Micro Cookbook, vol. 1", Lancaster..................($15.95)  $15.00
    "Beneath Apple DOS", Worth & Lechner.................($19.95)  $18.00
    "Bag of Tricks", Worth & Lechner, with diskette......($39.95)  $36.00
    "Apple Graphics & Arcade Game Design", Stanton.......($19.95)  $18.00
    "Assembly Lines: The Book", Roger Wagner.............($19.95)  $18.00
    "What's Where in the Apple", Second Edition..........($24.95)  $23.00
    "6502 Assembly Language Programming", Leventhal......($16.99)  $16.00
    "6502 Subroutines", Leventhal........................($12.99)  $12.00
    "MICRO on the Apple--1", includes diskette...........($24.95)  $23.00
    "MICRO on the Apple--2", includes diskette...........($24.95)  $23.00
    "MICRO on the Apple--3", includes diskette...........($24.95)  $23.00
```

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We take Master Charge, VISA and American Express ***

As explained above, the JSR VECTOR is in effect a JSR HANDLER. The Header Part 2 code takes the values in the A and Y registers and adds them to an address which it obtains from the stack to obtain the address of SUBR. It then places this address in INDEX ($5E,5F) and executes "JMP (INDEX)".

An internal JMP, from one part of your routine to another, is performed in a similar manner. Suppose you wish to JMP from HERE to THERE. It is done as follows:

```
HERE   LDA #THERE-HERE-7 low byte of offset
       LDY /THERE-HERE-7 high byte of offset
       TSX
       JSR $FF58
       JMP VECTOR
```

Since we are (in effect) performing a JMP, rather than a JSR, we do a JMP VECTOR rather than a JSR VECTOR. The other difference is that we have a JSR $FF58 following the TSX.

Clearly the sequence of instructions which allows you to perform a JMP or a JSR could be coded as a macro. The macros to use are shown in the accompanying program listing. By using macros an internal JMP or JSR can be performed with a single macro instruction bearing a very close resemblance to a real JSR or JMP instruction.

The following program, which consists of the Header Routine plus a demonstration routine, can be assembled to disk using the .TF directive. It can then be BRUN at any address and it will function properly. Thus it is relocatable, despite the fact that there are (in effect) an internal JMP and two internal JSRs performed.

When performing an internal JSR or JMP using my techniques, it is not possible to pass values in the registers, since these are required to pass information to the HANDLER routine. Nor is it advisable to try to pass parameters on the stack, even though the HANDLER routine does not change the value of the stack pointer. Better is to deposit values in memory and retrieve them after the transition has been made.

The HANDLER routine passes control to the requested part of your routine using a JMP indirect. (INDEX at $5E,5F, has been used in the accompanying program, but any other address would do as well, provided that it does not cross a page boundary.) This means that the section of your routine to which control is passed (whether or not it is a subroutine) may find its own location by inspecting the contents of the location used for the JMP indirect. This feature of this technique is also illustrated in the accompanying program, in the PRINT.MESSAGE subroutine.

The use of internal data blocks is something not normally possible in a relocatable routine, but it can be done if the techniques shown here are used.

This method of performing internal JSRs and JMPs in a

relocatable routine may be simplified if the routine is intended to function as a subroutine appended to an Applesoft program. If the subroutine is appended using my utility the Routine Machine (available from Southwestern Data Systems), then it is not necessary to include the 47-byte Header Routine. Internal JMPs and JSRs can still be performed exactly as described above, except that the address of VECTOR must be $3F5-$3F7. This technique is not described in the documentation to the Routine Machine. A full explanation may be found in the Appendix to the documentation accompanying Ampersoft Program Library, Volume 4 (also available from Southwestern Data Systems).

```
                    1000        .TF B.MEYER.1
                    1010 *--------------------------------
                    1020 *    SETUP AND HANDLER ROUTINES
                    1030 *    TO ALLOW INTERNAL JSRS AND
                    1040 *    JMPS IN A RELOCATABLE MACHINE
                    1050 *    LANGUAGE ROUTINE
                    1060 *
                    1070 *    BY PETER MEYER, 11/3/82
                    1080 *--------------------------------
                    1090 *    LOCATIONS
                    1100
 005E-              1110 INDEX      .EQ $5E,5F
 0100-              1120 STACK      .EQ $100 - $1FF
 03F5-              1130 VECTOR     .EQ $3F5 - $3F7
                    1140 *--------------------------------
                    1150 *    MACRO DEFINITIONS
                    1160
                    1170        .MA JSR
                    1180 :1     LDA #]1-:1-7
                    1190        LDY /]1-:1-7
                    1200        TSX
                    1210        JSR VECTOR
                    1220        .EM
                    1230
                    1240        .MA JMP
                    1250 :1     LDA #]1-:1-7
                    1260        LDY /]1-:1-7
                    1270        TSX
                    1280        JSR $FF58
                    1290        JMP VECTOR
                    1300        .EM
                    1310 *--------------------------------
                    1320 *    HEADER PART 1
                    1330
 0800- 20 58 FF     1340 SETUP  JSR $FF58      FIND OURSELVES
 0803- BA           1350        TSX
 0804- 18           1360        CLC
 0805- A9 1A        1370        LDA #HANDLER-SETUP-2
 0807- 7D FF 00     1380        .DA #$7D,STACK-1   FORCE ABS,X MODE
 080A- 8D F6 03     1390        STA VECTOR+1
                    1400
 080D- A9 00        1410        LDA /HANDLER-SETUP-2
 080F- 7D 00 01     1420        ADC STACK,X
 0812- 8D F7 03     1430        STA VECTOR+2
                    1440
 0815- A9 4C        1450        LDA #$4C       "JMP"
 0817- 8D F5 03     1460        STA VECTOR
 081A- D0 0F        1470        BNE MAIN.ROUTINE   ALWAYS
                    1480 *--------------------------------
                    1490 *    HEADER PART 2
                    1500
                    1510 HANDLER
                    1520
                    1530 *    ON ENTRY A,Y HOLD OFFSET
                    1540 *    FOR JMP OR JSR FROM ROUTINE
                    1550 *    X IS STACK POINTER FROM BEFORE LAST JSR
                    1560
 081C- 18           1570        CLC
 081D- 7D FF 00     1580        .DA #$7D,STACK-1   FORCE ABS,X MODE
 0820- 85 5E        1590        STA INDEX
```

```
0822- 98          1600        TYA
0823- 7D 00 01    1610        ADC STACK,X
0826- 85 5F       1620        STA INDEX+1
0828- 6C 5E 00    1630        JMP (INDEX)
                  1640 *-------------------------------
                  1650 *      MAIN ROUTINE, FOR EXAMPLE
                  1660 *-------------------------------
0006-             1670 MSG    .EQ $06 AND $07
0024-             1680 CH     .EQ $24
0025-             1690 CV     .EQ $25
0032-             1700 INVFLG .EQ $32
003C-             1710 COUNT  .EQ $3C
FB39-             1720 SETTXT .EQ $FB39
FC24-             1730 VTABZ  .EQ $FC24
FC58-             1740 HOME   .EQ $FC58
FDED-             1750 COUT   .EQ $FDED
                  1760 *-------------------------------
                  1770 MAIN.ROUTINE
082B- 20 39 FB    1780        JSR SETTXT
082E- 20 58 FC    1790        JSR HOME
                  1800 MAIN.LOOP
0831- A9 BE       1810        LDA #190
0833- 85 3C       1820        STA COUNT
0835- A9 38       1830 .1     LDA #AALQT-PRINT.MESSAGE
0837- 85 06       1840        STA MSG
0839- A9 01       1850        LDA /AALQT-PRINT.MESSAGE
083B- 85 07       1860        STA MSG+1
083D-             1870        >JSR PRINT.MESSAGE
083D- A9 20       0000> :1     LDA #PRINT.MESSAGE-:1-7
083F- A0 00       0000>        LDY /PRINT.MESSAGE-:1-7
0841- BA          0000>        TSX
0842- 20 F5 03    0000>        JSR VECTOR
0845- C6 3C       1880        DEC COUNT
0847- D0 EC       1890        BNE .1
0849- A9 3C       1900        LDA #LONGQT-PRINT.MESSAGE
084B- 85 06       1910        STA MSG
084D- A9 01       1920        LDA /LONGQT-PRINT.MESSAGE
084F- 85 07       1930        STA MSG+1
0851-             1940        >JSR PRINT.MESSAGE
0851- A9 0C       0000> :1     LDA #PRINT.MESSAGE-:1-7
0853- A0 00       0000>        LDY /PRINT.MESSAGE-:1-7
0855- BA          0000>        TSX
0856- 20 F5 03    0000>        JSR VECTOR
0859-             1950        >JMP FORWRD
0859- A9 22       0000> :1     LDA #FORWRD-:1-7
085B- A0 01       0000>        LDY /FORWRD-:1-7
085D- BA          0000>        TSX
085E- 20 58 FF    0000>        JSR $FF58
0861- 4C F5 03    0000>        JMP VECTOR
                  1960
                  1970 *-------------------------------
                  1980 PRINT.MESSAGE
0864- 18          1990        CLC
0865- A5 06       2000        LDA MSG       CHANGE RELATIVE ADDRESS TO
0867- 65 5E       2010        ADC INDEX     AN ABSOLUTE ADDRESS, BY
0869- 85 06       2020        STA MSG       ADDING THE OFFSET
086B- A5 07       2030        LDA MSG+1
086D- 65 5F       2040        ADC INDEX+1
086F- 85 07       2050        STA MSG+1
0871- A0 00       2060        LDY #0        POINT AT FIRST CHAR OF MSG
0873- B1 06       2070 .1     LDA (MSG),Y   GET NEXT CHAR OF MSG
0875- 30 08       2080        BMI .2        IT IS LAST CHAR
0877- 09 80       2090        ORA #$80      MAKE APPLE VIDEO FORM
0879- 20 ED FD    2100        JSR COUT      PRINT IT
087C- C8          2110        INY           ADVANCE POINTER
087D- D0 F4       2120        BNE .1        ...ALWAYS
087F- 4C ED FD    2130 .2     JMP COUT      PRINT AND RETURN
                  2140 *-------------------------------
                  2150 *    256 BYTES TO JUMP OVER, JUST FOR ILLUSTRATION
                  2160
0882-             2170        .BS $100
                  2180 *-------------------------------
                  2190 *    TOGGLE INVERSE FLAG, AND HOME CURSOR
                  2200
0982- A5 32       2210 FORWRD LDA INVFLG
0984- 49 C0       2220        EOR #$C0
0986- 85 32       2230        STA INVFLG
0988- A9 00       2240        LDA #0
098A- 85 24       2250        STA CH
098C- 85 25       2260        STA CV
```

```
098E- 20 24 FC 2270          JSR VTABZ
0991-          2280          >JMP MAIN.LOOP
0991- A9 99    0000> :1       LDA #MAIN.LOOP-:1-7
0993- A0 FE    0000>          LDY /MAIN.LOOP-:1-7
0995- BA       0000>          TSX
0996- 20 58 FF 0000>          JSR $FF58
0999- 4C F5 03 0000>          JMP VECTOR
               2290 *--------------------------------
099C- 41 41 4C
099F- A0       2300 AALQT   .AT /AAL /
09A0- 0D 0D    2310 LONGQT  .HS 0D0D
09A2- 20 41 20
09A5- 50 20 50
09A8- 20 4C 20
09AB- 45 20 20
09AE- 20 41 20
09B1- 53 20 53
09B4- 20 45 20
09B7- 4D 20 42
09BA- 20 4C 20
09BD- 59 20 20
09C0- 20 4C 20
09C3- 49 20 4E
09C6- 20 45 20 2320          .AS / A P P L E   A S S E M B L Y   L I N E /
09C9- 0D 02    2330          .HS 0D02
09CB- 20 20 53
09CE- 20 2D 20
09D1- 43 20 20
09D4- 20 53 20
09D7- 4F 20 46
09DA- 20 54 20
09DD- 57 20 41
09E0- 20 52 20
09E3- 45 20 20
09E6- 20 43 20
09E9- 4F 20 52
09EC- 20 50 20
09EF- 2E 20 A0 2340          .AT /  S - C   S O F T W A R E   C O R P   /
```

Add Bit-Control to Apple Monitor..........Bob Sander-Cederlof

The other day someone sent me a disk with an article for AAL on it as a binary file. The codes in the file were all ASCII characters, but they were nevertheless not compatible with any word processors I had around.

All blanks were coded as $A0 (hi-bit on), and all other characters were coded in the range $00-$7F (hi-bit off). Otherwise, everything was compatible with my favorite word processor (the one I am still in the process of finishing).

I need to have all the hi-bits set to one in the file, or in the memory image after BLOADing the file. That's the motivation for the following neat enhancement to the Apple monitor.

The enhancement hooks in through the control-Y user command vector. By merely typing:

```
     *80FF<2000.3FFF^Y     (^Y means control-Y)
```

I set all the hi-bits between $2000 and $3FFF.

The "80FF" in the command line above is the magic part of this enhancement. The last two digits are ANDed with every byte in the specified range, clearing every bit which is zero in those two digits. By using $FF, no bits are cleared. Other values for these two digits will clear whatever bits you wish.

The first two digits are ORed into every byte in the specified range, setting every bit which is one in the two digits. $80 in my example sets the top bit in every byte.

The code is designed to be BRUN from a binary file, and it is completely relocatable. You can BRUN it anywhere in memory that you have room for 36 bytes. That is why the SETUP code is longer than the actual control-Y code!

The SETUP routine first discovers where in memory it is running, and then sets up the control-Y vector in page 3 to point at the BITS code. The discovery is done in the usual way, by JSRing to a guaranteed RTS in the monitor ROM, at $FF58. This leaves a return address just below the stack pointer. I pick up that address and add the difference between that and BITS to get the appropriate control-Y vector pointer.

Line 1200 needs a little explanation. My assembler automatically switches to page zero addressing mode if the address is less than $100. STACK-1 is less than $100, so "ADC STACK-1,X" would assemble as though I wrote "ADC $FF,X". Indexed addressing in page zero mode stays in page zero, wrapping around. If X=3, "ADC $FF,X" would reference location $02 in page zero rather than $102. Therefore I had to use the ".DA #$7D" to force the assembler to use a full 16-bit address mode. (Some assemblers have a special way of forcing 16-bit addressing in cases like this; others require special marks to get zero-page modes.)

BITS itself is very straightforward code. The monitor leaves the starting address of the specified range in A1 ($3C,3D), the ending address in A2 ($3E,3F), and the mask in A4 ($42,43). The subroutine at $FCBA increments A1 and compares it to A2, leaving carry clear if the range is not yet complete.

```
                   *--------------------------------
                   *      MONITOR CTRL-Y COMMAND
                   *
                   *      TO SET AND CLEAR ANY COMBINATION
                   *      OF BITS IN A RANGE OF MEMORY
                   *
                   *      *MASK<ADR1.ADR2Y   (WHERE Y MEANS CTRL-Y)
                   *
                   *      MASK = XXYY  BITS = 0 IN YY WILL BE CLEARED
                   *                   BITS = 1 IN XX WILL BE SET
                   *--------------------------------
003C-              A1     .EQ $3C
0042-              A4L    .EQ $42
0043-              A4H    .EQ $43
0100-              STACK  .EQ $100
                   *--------------------------------
0800- 20 58 FF     SETUP  JSR $FF58     FIND SELF FIRST
0803- BA                  TSX
0804- 18                  CLC
0805- A9 14               LDA #BITS-SETUP-2
0807- 7D FF 00            .DA #$7D,STACK-1     FORCE ABS,X MODE
080A- 8D F9 03            STA $3F9          MONITOR CTRL-Y VECTOR
080D- A9 00               LDA /BITS-SETUP-2
080F- 7D 00 01            ADC STACK,X
0812- 8D FA 03            STA $3FA
0815- 60                  RTS
                   *--------------------------------
0816- B1 3C        BITS   LDA (A1),Y    GET NEXT BYTE IN SPECIFIED RANGE
0818- 25 42               AND A4L       CLEAR BITS USING LO-BYTE OF MASK
081A- 05 43               ORA A4H       SET BITS FROM HI-BYTE OF MASK
081C- 91 3C               STA (A1),Y    STORE MODIFIED BYTE
081E- 20 BA FC            JSR $FCBA     ADVANCE POINTER
0821- 90 F3               BCC BITS      MORE IN RANGE
0823- 60                  RTS           FINISHED
                   *--------------------------------
```

Assembly Listings on TEXT Files............Bob Sander-Cederlof

Today Jules Gilder called, asking for patches to allow sending the assembly listing to a TEXT file. He is in the process of writing a book, and wanted the listings on TEXT files so they could be automatically typeset.

My first response was a familiar one: "It is a very difficult problem, because of the interaction of .IN and .TF files."

"But I don't need .TF or .IN files!", he swiftly parried.

Suddenly in a flash of insight I saw that it could be EASILY done. All that is needed is to patch the .TF directive so that it opens a TEXT file, but does not set the TF-flag. Then all listing output will go to the text file, and the object code will go to the current origin or target address.

Here are the patches for the motherboard version:

```
:$2998:A5 60 D0 03 20 6A 2A 4C 04 1F
```

And the language card version:

```
:$C083 C083 EAE4:A5 60 D0 03 20 B6 EB 4C 50 E0
```

Note that the two "C083's" above write-enable the language card so the patches will be effective.

After the patches are installed, a .TF directive will direct the listing to your text file. Here is an example:

```
       .TF T.LISTING
SAMPLE LDA #3
       STA $06
       RTS
```

Just remember that the normal use of .TF is not available when this patch is in place; also that .IN should not be used. Using .IN would turn off the listing output, directing it back to the screen.

Quickie No. 1.............................Bob Sander-Cederlof

To merge selected bits from one byte with the rest of the bits of another byte:

```
  Code            Example
LDA MASK          00011111
EOR #$FF          11100000
ORA BYTE1         111xxxxx
STA TEMP
LDA BYTE2         yyyzzzzz
ORA MASK          yyy11111
AND TEMP          yyyxxxxx
```

Add a New Feature to ES-CAPE......................Bill Linn

Since most of the owners of ES-CAPE also subscribe to the Apple Assembly Line, I thought I would pass on some neat patches here.

The automatic line numbering feature in ES-CAPE can be enhanced by the following patches, which make the numbers track whatever you type. With these patches, each time an Applesoft line is changed via hitting return or via a CHANGE command, that line number plus the current increment becomes the next automatic line number to be generated when the space bar is pressed. (Now ES-CAPE works more like the S-C Macro Assembler.)

Here are the patches for the mother-board version:

```
LOAD ES-CAPE
CALL -151
E44:69 00 EA
102E:4C 51 9B
1C51:A5 51 8D 34 9B 18 A5 50 6D 35 9B 8D 33 9B
:90 03 EE 34 9B A5 25 F0 02 C6 25 4C 34 8F
3D0G
SAVE ES-CAPE REVISED
```

Here are the patches for the RAM-card version:

```
LOAD ES-CAPE LC
CALL -151
E41:69 00 EA
102B:4C 60 E1
1B60:A5 51 8D 51 E1 18 A5 50 6D 52 E1 8D 50 E1
:90 03 EE 51 E1 A5 25 F0 02 C6 25 4C 31 D6
3D0G
SAVE ES-CAPE LC REVISED
```

I believe these patches make ES-CAPE even easier to use. If any of you still have not upgraded your AED II copies to ES-CAPE, send me $10 and your old disk and I'll return a new version and the great new manual.

I am continuing to work on ES-CAPE, hoping for a new version around the middle of next year. What new features would you like? The main ones we have in mind are 80-column support, renumber, and merge. If you have some good ideas, drop me a line at 3199 Hammock Creek, Lithonia, GA 30058.

Quickie No. 2............................Bob Sander-Cederlof

To test a byte in memory without disturbing any registers:

```
INC BYTE
DEC BYTE        Restore value, and test against zero
BEQ
```

Applesoft Source, Completely Commented.....Bob Sander-Cederlof

For several years I have been working on this. I even bought an assembler from another company just to get the promised source code of Applesoft that came in the package, but I was very disappointed. (No complaints, it was intended as a "freebie" with their assembler.) I wanted LOTS of comments, MEANINGFUL labels, and I wanted it in the format of the S-C Macro Assembler.

Now I have it. And you can have a copy, on two diskettes, for only $50. It comes in an encrypted form, with a driving program which creates the source code files on your machine with the aid of the Applesoft image in ROM or RAM. The encryption is meant to protect the interests of Apple and Microsoft.

You need two disk drives to assemble Applesoft, a printer to get a permanent listing, and of course you need the S-C Macro Assembler. The source code is split into more than a dozen source files, assembled together using a control file full of .IN directives. After assembling and printing, you will have well over 100 pages of the best documentation of Applesoft internals available anywhere.

In the process of writing the comments, I discovered some very interesting bugs. Some have been published before, and others I have never heard of. Just for fun, try this:

```
]A%=-32768          (you get an error message, correctly)
]A%=-32768.00049         (No error message!)
]PRINT A%                (You get 32767!)
```

Or open your disk drive doors, just in case, and type:

```
]LIST 440311
```

That last one can be disastrous! Any use of a six-digit line number (illegal, but not caught by Applesoft) between 437760 and 440319 will cause a branch to a totally incorrect place. For example, GOTO 440311 branches to $22D9!

The causes of these and other interesting phenomena, as well as some suggested improvements resulting in smaller/faster code, are documented in my comments.

Quickie No. 3.............................Bob Sander-Cederlof

To shift a two-byte value right one bit with sign extension:

```
LDA HI.BYTE
ASL             SIGN BIT INTO CARRY
ROR HI.BYTE     HI BYTE RIGHT ONE, CARRY (SIGN) INTO BIT 7
ROR LO.BYTE
```

New Enhanced 6502 Nearly Here!............Bob Sander-Cederlof

Nigel Nathan from Micro-Mixedware in Stow, MA, sent me some copies of reference material for the new 65C02 chip. This is the enhanced CMOS version, soon to be available from GTE and Rockwell.

Nigel's interest was that I might produced an enhanced S-C Macro Assembler to accommodate the new opcodes and addressing modes. I not only might...I did it right away! It is ready now, although you may have some difficulty getting the chips for a few more months.

Rockwell is sampling the 65C02 now, and scheduled for production in February. Rockwell is also readying an entire family of CMOS products to go with the 65C02, including 2Kx8 CMOS static RAM and multi-byte parallel interfaces.

The 65C02 is expected to be plug-compatible with the 6502 in your Apple II. In fact, Cliff Whitaker of Rockwell told me that the first chips they made were tested by plugging them into Apples. Hopefully you will be able to simply plug them in and start using the new opcodes. If true, then I will probably become a source for these chips.

What enhancements did they make? According to the GTE document, some "bugs" in the 6502 design were corrected:

* Indexed addressing across a page boundary no longer causes a false read at an invalid address.

* Invalid opcodes are now all NOPs, rather than doing exotic things such as I described in the March 1981 AAL.

* JMP indirect at a page boundary now operates correctly, at a cost of one additional cycle.

* Read/modify/write opcodes (like INC, DEC, and the shifts) now perform two reads and one write cycle rather than one read and two writes.

* The D-status bit is now set to binary mode (D=0) by reset; it used to be indeterminate.

* The N-, V-, and Z-status bits are now valid after ADC or SBC in decimal mode (D=1); they used to be invalid, requiring special tests. The cost is one additional cycle for all ADCs and SBCs in decimal mode.

* An interrupt after fetch of a BRK opcode defers to the BRK. It used to cause the BRK to be ignored.

The Rockwell literature makes reference to the following new opcodes, or new addressing modes for old opcodes:

```
     80     BRA rel      Branch Always

     12     ORA (zp)     )
     32     AND (zp)     )
     52     EOR (zp)     )  new addressing mode:
     72     ADC (zp)     )
     92     STA (zp)     )  zero-page indirect
     B2     LDA (zp)     )
     D2     CMP (zp)     )  without indexing
     F2     SBC (zp)     )

     04     TSB zp       Test and set bits
     14     TRB zp       Test and reset bits
     34     BIT zp,X     new addressing mode for BIT
     64     STZ zp       Store Zero
     74     STZ zp,X       "    "

     07-77     RMB b,zp     Reset bit b in zp
     87-F7     SMB b,zp     Set bit b in zp

     89     BIT imm      new addressing mode for BIT

     1A     INC          Increment A-register
     3A     DEC          Decrement A-register
     5A     PHY          Push Y
     7A     PLY          Pull Y
     DA     PHX          Push X
     FA     PLX          Pull X

     0C     TSB abs      Test and set bits
     1C     TRB abs      Test and reset bits
     3C     BIT abs,X    new addressing mode for BIT
     7C     JMP (abs),X     new addressing mode for JMP
     9C     STZ abs      Store zero

     9E     STZ abs,X    Store zero

     0F-7F     BBR b,zp,rel     Branch if bit b in zp is zero
     8F-FF     BBS b,zp,rel     Branch if bit b in zp is one
```

Let your imagination run wild with all the great ways to use these new opcodes! If you feel the need for the ability to assemble them now, the Cross Assembler upgrade for the 65C02 is available for $20 to subscribers of the Apple Assembly Line who already own the S-C Macro Assembler.

HardCore Computing is a quarterly magazine.

Toggling Upper/Lower Case in the S-C Macro Assembler.........

Steven Mann
Psychology Dept.
Univ. of So. Dakota
P. O. Box 7187
Grand Forks, ND 58202

I have made the necessary modifications to the assembler (and to my Apple) that allow me to enter lower case characters in my source programs, but have found that I like to have all upper case in certain sections (the labels and opcodes) and mixed (mostly lower) case only in the comment field. In order to do accommodate this most effectively, I wanted to be able to toggle back and forth from upper to lower case while I was entering my source code.

The simplest solution for me was to patch the assembler to accept one of the escape key sequences as an upper/lower case toggle. From back issues of AAL I was able to determine that a table of address vectors for the escape keys A-L is maintained from $1467 thru $1482 ($D467 thru $D482 in the language card version). Each two-byte entry is the address-1 (low byte first) of the routine that will handle that particular escape sequence.

Certain of the sequences are already taken (e.g. ESC L loads a disk file; ESC I,J,K, and M move the cursor, etc.) Since I don't use the ESC A,B,C,D cursor moves, I selected the ESC A sequence as the code for the case toggle. I also used ESC C for "CATALOG", as suggested by Bill Morgan some months back in these pages.

Implementation of the toggle is accomplished with the following patches to the HELLO program (for the RAM version of the assembler.) First, line 50 should be changed to:

```
50 PRINT A: IF A=1 THEN PRINT CHR$(4)"BLOAD S-C.ASM.MACRO"
   :GOSUB 200: CALL 4096
```

The subroutine at 200 is as follows:

```
197 REM
198 REM ESC A TOGGLES UP/LOW CASE
199 REM
200 POKE 5229,109:POKE 5230,165
210 FOR I=1 TO 9:READ J:POKE 48350+I,J:NEXT
220 DATA 173,22,16,73,255,141,22,16,96
230 RETURN
240 REM
250 REM ROUTINE RESIDES AT $BCDF
260 REM
270 REM CODE IS AS FOLLOWS:
280 REM
290 REM     LDA $1016
300 REM     EOR #FF
310 REM     STA $1016
320 REM     RTS
330 REM
```

Note that I put the patch code at $BCDF, which is in an unused area inside DOS 3.3. If you have already used this area for other purposes, you can tack the patch on to the end of the assembler image instead.

The actual code is very simple. The upper/lower case flag is stored at $1016 and is either a $00 or a $FF (in binary all zeros or all ones.) Toggling the flag involves loading the current flag and EORing it with #$FF. This will cause all set bits to be cleared and all clear bits to be set, so the zeros become ones and the ones become zeros. Thus, an #$FF byte becomes a #$00 or a #$00 becomes an #$FF. The new flag value is then stored back in $1016.

For the language card version the program is basically the same, but slightly longer due to the need to first write enable the language card. The code looks like this:

```
PATCH   LDA $C083       Two of these in succession
        LDA $C083       write-enable the card
        LDA $D016       Get the flag
        EOR #$FF        Complement it
        STA $D016       Save the new flag
        LDA $C080       Write protect the card
        RTS
```

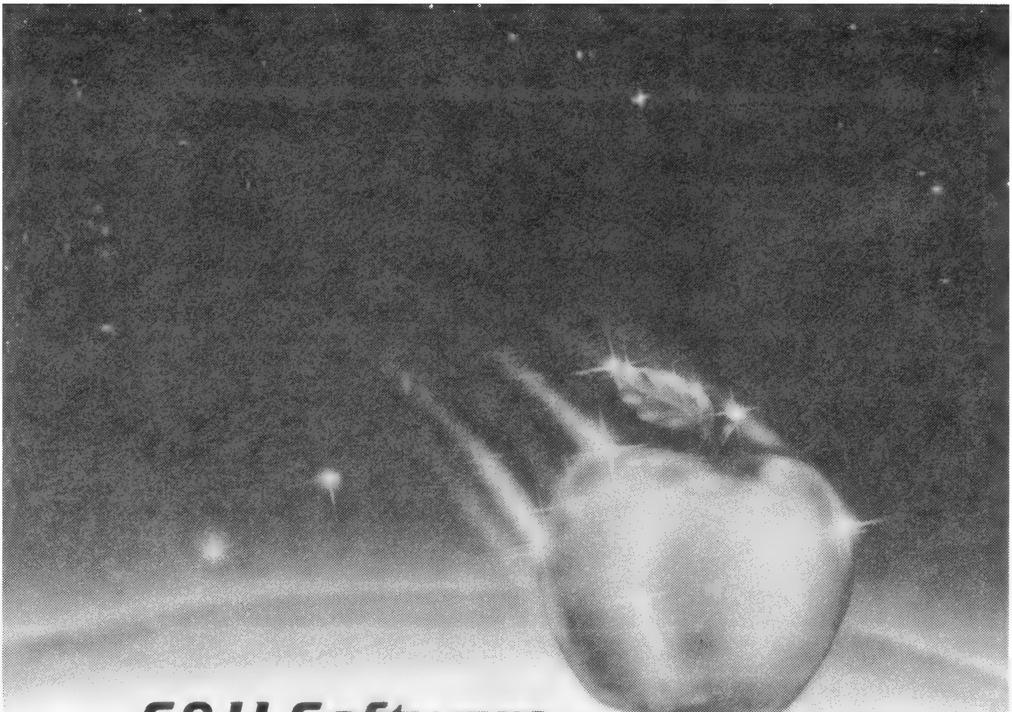
I put the code in the same place as in the RAM version ($BCDF) and put it into memory from the LOAD LCASM exec file which loads the assembler onto the card. Two lines need to be added to the file. Between lines 1070 and 1080 (assuming your version has not been modified) you need to place these two lines:

```
1072 D469:DE BC
1074 BCDF:AD 83 C0 AD 83 C0 AD 16 D0 49 FF 8D 16 D0 AD 80
C0 60
```

The first line places the address of the case toggle handler in the escape vector table and the second line contains the assembled source code listed above. If you are not sure how to modify the LOAD LCASM file see the step by step description given in the May 1982 AAL (page 3).

After you have made the patch, experiment with the toggle. One particularly valuable feature is that you can toggle the case within a single line as you enter the line. This means that you can enter the label and opcode in upper case, tab over to the comment field, toggle the case flag, and then enter your comment in lower case.

I have found using the case toggle to be easy while improving the appearance and readability of my source listings. The only problem I have encountered so far is that the flag can not be toggled from within the edit mode (either case can be used in the edit mode, but you can't change the case in the middle of editing.) If any of you find a way to add this to the assembler be sure to let me know.



S&H Software is Speeding up The Apple®!

The DOS Enhancer (TDE) works up to 500% faster than standard Apple DOS 3.3...$69.95

TIRED OF WAITING . . . for your programs to load or save? Then S&H's TDE, licensed by Apple, is the answer.

TDE program updates standard Apple DOS 3.3 disks — or creates copyable TDE disks — with TDE's QuickDOS and Quick-load features.

TDE's QuickDOS runs and saves BASIC and binary programs up to 5 times faster* and is completely compatible with standard Apple DOS 3.3 programs.

TDE "Quick-loads" the RAM card with FPBASIC/INTBASIC, QuickDOS or user program in 1.7 seconds at startup.

TDE "package" includes utility disk, training/support disk, step-by-step instruction manual, S&H's Supercat/menu, and multidrive "Quick-copy" program.

TDE system requirements: 48K Apple][or][+ ROM/RAM card, DOS 3.3 and one or more disk drives.

Here's what the critics say:

- John Mitchener of PEELINGS II: "The speed increase with **TDE** is awesome and is probably worth the price of the program alone, without all the other features ... AA rating."
- Val Golding, Editor of Call-A.P.P.L.E.: "(**TDE**) stands as a shining example of how utility and application programs take into account every possible system configuration."
- Clark Congleton of The Apple Orchard: "The Quick-load capabilities will make this package attractive to anyone who spends a lot of time at the keyboard."
- Chuck Carpenter of INFOWORLD: "Results in a disk that will boot — very fast — in any Apple system."

*To achieve speeds even faster than a hard-disk drive, c 320K Memory System

Amper-Sort/Merge (A-S/M II) works up to 1000% faster than even VisiCorp's VisiFile program...$69.95

A-S/M II is the fastest "file clerk" you've ever met. Of all the sort utilities developed to manage Apple][data files none does the job nearly so fast as A-S/M II!

A-S/M II's new features include S&H's superfast VisiFile index sort (callable from within VisiFile for effortless use), an equally fast S&H random access file index sort, and parameter file editing.

A-S/M II can sort/merge from one to five unsorted files into a single file of up to 125K in size per disk.

A-S/M II's "package" includes utility disk, training disk, step-by-step instruction manual, and S&H's new Supercat/menu.

A-S/M II's system requirements: 48K Apple][with ROM or RAM card or 48K Apple][+ with DOS 3.3 and Disk II.

Dealer inquiries invited

Apple is a registered trademark of Apple Computer, Inc.

S&H Software

Available from your dealer. **Mail Order**: Send checks to S&H Software, 58 Van Orden Rd., Harrington Park, NJ 07640, 201-768-3144

Credit Cards: Phone Cybertronics International at 212-532-3089 (Overseas Airmail: Add $3.00 postage and handling.)

Save Garbage by Emptying Arrays...........Bob Sander-Cederlof

As we all know, Applesoft programs which use a significant number of strings can appear to die for long periods of time while "garbage collection" is performed. Many techniques have been published to reduce the frequency of garbage collection, or reduce the amount of garbage to be collected, or to speed up the collector.

Randy Wiggington published a much faster garbage collector in "Call-A.P.P.L.E.", January, 1981, pages 40-45. The source code is available in S-C format on the Dallas Apple Corps disk of the month for March, 1981. (Copies available for $10 from me.) Randy's speed improvement is gained by searching for the highest 16 strings in memory at once, rather than just the highest one string. It is much faster, but not the fastest. The time for collection still varies quadratically with the number of strings in use.

Cornelius Bongers wrote the fastest collector I have seen. It was published in "MICRO -- The 6502/6809 Journal", August, 1982, pages 90-97. Corny's method is very straightforward, and has the advantage that execution time varies linearly with the number of strings in use. His method also has the disadvantage that it does not work if any strings contain any characters with the high bit on. (Applesoft normally does not generate any strings with high-bit-set-characters, but you can do it with oddball code.) I typed in the program from MICRO, made a few changes here and there, and found it to be lightning fast.

I installed the linear method in a client's program, and his garbage collection time went from 100 seconds after printing every four lines, to only 1/4 second. Other changes, such as the one described below, cut the interval of collection to once every 12 lines.

It so happens that strings which are empty do not increase the garbage collection time. Many times in Applesoft programs a string array is filled with data from a disk file, processed, and then filled with fresh data, and so on. By emptying the array before each pass at reading the disk file, the number of active strings can be greatly reduced.

One of my programs was like this: the one that prints your mailing label so that the AAL gets to you every month. Before reading each file (the list is divided into about 12 different files, based on zip code and other factors), I wrote a loop that set each string in the array to a null string, and then forced garbage collection:

```
     FOR I = 1 TO NH   A$(I)=""   NEXT   F = FRE(0)
```

The only problem with this was that the loop takes so long! About ten seconds, anyway, enought to try my patience.

All the above motivated me to write the following little subroutine, which nulls out (or empties) a string array. Bongers included an array eraser in his article, which

completely released an array; however, that requires re-DIMming the array each time. My program is faster in my case, and it was fun to write.

The program is listed below with the origin set at $300, so "CALL 768,arrayname" will activate it. It happens to be fully relocatable, so you can load it anywhere in memory you wish. You could easily add it to your Applesoft program with Amper-Magic or Amperware or The Routine Machine.

I used three subroutines inside the Applesoft ROMs. CHKCOM gives SYNTAX ERROR unless the next character is a comma. I use it to check for the comma that separates the call address from the array name. CHKSTR checks to make sure that the last-parsed variable is a string variable, and gives TYPE MISMATCH if not. GETARYPT scans an array name and returns the address of the start of the array in variable space.

If you look at page 137 of your Applesoft reference manual, you will see in the third column a picture of a string array. (Notice first the error: the signs of the first and second bytes of the string name are just the reverse of what the book says.) My program looks at the number of dimensions to determine the size of the preamble (the number of bytes before you get to the actual string pointers).

I use the preamble size to compute the starting address of the string pointers, and the number of bytes of string pointers that there are. Then a tight little loop stores zeros on top of all the descriptors.

The Applesoft program below illustrates the CLEAR subroutine in action.

```
100  DIM A$(25),B$(5)
105  FOR I = 0 TO 5:B$(I) = "ABCD": NEXT
110  FOR I = 0 TO 25:N =  INT ( RND (1) * 5) + 1
120  FOR J = 1 TO N:A$(I) = A$(I) +  CHR$ ( RND (1) * 26 + 65): NE
     : PRINT  LEN (A$(I))":"A$(I),
130  NEXT : PRINT : PRINT
140  CALL 768,A$
150  FOR I = 0 TO 25: PRINT  LEN (A$(I))":"A$(I),: NEXT
155  PRINT : PRINT
160  FOR I = 0 TO 5: PRINT B$(I),: NEXT
170  GOTO 110
```

QUICKTRACE

relocatable program traces and displays the actual machine operations, *while* it is running without interfering with those operations. Look at these *FEATURES:*

***Single-Step** mode displays the last instruction, next instruction, registers, flags, stack contents, and six user-definable memory locations.*

***Trace** mode gives a running display of the Single-Step information and can be made to stop upon encountering any of nine user-definable conditions.*

***Background** mode permits tracing with no display until it is desired. Debugged routines run at near normal speed until one of the stopping conditions is met, which causes the program to return to Single-Step.*

***QUICKTRACE** allows changes to the stack, registers, stopping conditions, addresses to be displayed, and output destinations for all this information. All this can be done in Single-Step mode while running.*

***Two optional display formats** can show a sequence of operations at once. Usually, the information is given in four lines at the bottom of the screen.*

***QUICKTRACE** is completely transparent to the program being traced. It will not interfere with the stack, program, or I/O.*

***QUICKTRACE** is relocatable to any free part of memory. Its output can be sent to any slot or to the screen.*

***QUICKTRACE** is completely compatible with programs using Applesoft and Integer BASICs, graphics, and DOS. (Time dependent DOS operations can be bypassed.) It will display the graphics on the screen while **QUICKTRACE** is alive.*

***QUICKTRACE** is a beautiful way to show the incredibly complex sequence of operations that a computer goes through in executing a program*

QUICKTRACE *$50*

Is a trademark of Anthro-Digital, Inc.
Copyright © 1981
Written by John Rogers

See these programs at participating Computerland and other line computer stores.

Anthro - Digital Software, Inc.
P.O. Box 1385 Pittsfield, MA 01202

```
                1000 *--------------------------------
                1010 *      CLEAR STRING ARRAY
                1020 *--------------------------------
F7D9-           1030 GETARYPT    .EQ $F7D9
DD6C-           1040 CHKSTR      .EQ $DD6C
DEBE-           1050 CHKCOM      .EQ $DEBE
                1060 *--------------------------------
009D-           1070 FAC    .EQ $9D
009B-           1080 LOWTR  .EQ $9B,9C
                1085 *--------------------------------
                1090        .OR $300
                1100 *--------------------------------
                1110 CLEAR
0300- 20 BE DE  1120        JSR CHKCOM
0303- 20 D9 F7  1130        JSR GETARYPT
0306- 20 6C DD  1140        JSR CHKSTR
0309- A0 04     1150        LDY #4        COMPUTE SIZE OF PREAMBLE
030B- B1 9B     1160        LDA (LOWTR),Y      # OF DIMS
030D- 0A        1170        ASL                *2, CLEAR CARRY
030E- 69 05     1180        ADC #5             +5
0310- 85 9D     1190        STA FAC       SAVE PREAMBLE SIZE
0312- A0 02     1200        LDY #2        POINT AT OFFSET
0314- 38        1210        SEC           COMPUTE ARRAY LENGTH
0315- B1 9B     1220        LDA (LOWTR),Y
0317- E5 9D     1230        SBC FAC
0319- 48        1240        PHA           # BYTES IN PARTIAL PAGE
031A- C8        1250        INY
031B- B1 9B     1260        LDA (LOWTR),Y
031D- E9 00     1270        SBC #0
031F- AA        1280        TAX           # WHOLE PAGES
0320- 18        1290        CLC           POINT AT BEGINNING OF DATA
0321- A5 9B     1300        LDA LOWTR
0323- 65 9D     1310        ADC FAC
0325- 85 9B     1320        STA LOWTR
0327- D0 02     1330        BNE .2
0329- E6 9C     1340        INC LOWTR+1
032B- A0 00     1350 .2     LDY #0
032D- 8A        1360        TXA           # WHOLE PAGES
032E- F0 0B     1370        BEQ .4
0330- 98        1380        TYA           SET A=0
0331- 91 9B     1390 .3     STA (LOWTR),Y
0333- 88        1400        DEY
0334- D0 FB     1410        BNE .3
0336- E6 9C     1420        INC LOWTR+1
0338- CA        1430        DEX           COUNT WHOLE PAGES
0339- D0 F6     1440        BNE .3
033B- 68        1450 .4     PLA
033C- F0 08     1460        BEQ .6        FINISHED
033E- A8        1470        TAY
033F- A9 00     1480        LDA #0
0341- 88        1490 .5     DEY
0342- 91 9B     1500        STA (LOWTR),Y
0344- D0 FB     1510        BNE .5
0346- 60        1520 .6     RTS
```

Splitting Strings to Fit Your Display......Bob Sander-Cederlof

Printing text on the screen, or even on a printer, is not as easy at it ought to be. The problem is splitting words at the right margin. Word processors handle it nicely, but what do you do in an Applesoft program?

You might write a subroutine, in Applesoft, which looks for the first space between words before a specified column position. The subroutine could split a string at the space into two substrings: one containing the next display line, the other the remainder of the original string.

You might. But believe me, it builds up a lot of garbage strings and takes a long time to execute. If you like the general approach, you might try coding the subroutine in assembly language. You can avoid garbage build-up and save lots of running time, so it is probably worth the effort. Especially since I already wrote the program for you!

The program is written to be called from an ampersand parser like the one in last month's article on REPEAT/UNTIL. Or, you can use it with Amper-Magic, Amperware, The Routine Machine, etc. It is fully relocatable, having no internal data or JMP/JSR addresses. I set the origin to $300, but it can be loaded and used anywhere without re-assembly.

Here is an Applesoft program to show how to call SPLIT:

```
100  POKE 1014,0: POKE 1015,3
120  FOR N = 5 TO 40 STEP 3: GOSUB 1000: NEXT : END
1000 A$ = "NOW IS THE TIME FOR ALL GOOD MEN TO COME
     TO THE AID OF THEIR PARTY."
1005  & ,A$,B$,N
1010  PRINT B$
1020  IF A$ < > "" THEN 1005
1025  PRINT
1030  RETURN
```

Call SPLIT with three parameters. The first (A$ above) is the source string, which will be split. After SPLITting, the remainder string will be left in A$.

The second parameter, B$ above, will receive the left part, including the last complete word, up to N (the 3rd parameter) characters. If there is no space in the left N characters, exactly N characters will be split.

Here are some of the printouts from the test program:

```
N=5            N=11           N=20
-----          -----------    --------------------
NOW            NOW IS THE     NOW IS THE TIME FOR
IS             TIME FOR       ALL GOOD MEN TO COME
THE            ALL GOOD       TO THE AID OF THEIR
TIME           MEN TO COME    PARTY.
   etc.        ...etc.
```

```
                         1000 *--------------------------------
                         1010 *        & SPLIT,A$,B$,W
                         1020 *
                         1030 *        A$ -- SOURCE STRING
                         1040 *        W  -- MAXIMUM WIDTH OF SPLIT
                         1050 *
                         1060 *        B$ -- LEFT W CHARS OF A$
                         1070 *        A$ -- REST OF A$
                         1080 *
                         1090 *--------------------------------
                         1100        .OR $300
                         1110        .TF B.SPLIT
                         1120 *--------------------------------
0050-                    1130 LINNUM      .EQ $50,51
0006-                    1140 DPTRA       .EQ $06,07
0008-                    1150 DPTRB       .EQ $08,09
00FE-                    1160 SPTRA       .EQ $FE,FF
                         1170 *--------------------------------
DEBE-                    1180 AS.CHKCOM   .EQ $DEBE
DFE3-                    1190 AS.PTRGET   .EQ $DFE3
E752-                    1200 AS.GETADR   .EQ $E752
DD67-                    1210 AS.FRMNUM   .EQ $DD67
                         1220 *--------------------------------
0300- 20 BE DE           1230 SPLIT  JSR AS.CHKCOM     GET COMMA
0303- 20 E3 DF           1240        JSR AS.PTRGET     GET SOURCE STRING
0306- 85 06              1250        STA DPTRA
0308- 84 07              1260        STY DPTRA+1
030A- 20 BE DE           1270        JSR AS.CHKCOM     ANOTHER COMMA
030D- 20 E3 DF           1280        JSR AS.PTRGET     GET TARGET STRING
0310- 85 08              1290        STA DPTRB
0312- 84 09              1300        STY DPTRB+1
0314- 20 BE DE           1310        JSR AS.CHKCOM     ANOTHER COMMA
0317- 20 67 DD           1320        JSR AS.FRMNUM
031A- 20 52 E7           1330        JSR AS.GETADR     GET MAXIMUM WIDTH
031D- A0 02              1340        LDY #2
031F- B1 06              1350        LDA (DPTRA),Y
0321- 85 FF              1360        STA SPTRA+1
0323- 91 08              1370        STA (DPTRB),Y
0325- 88                 1380        DEY
0326- B1 06              1390        LDA (DPTRA),Y
0328- 85 FE              1400        STA SPTRA
```

```
032A- 91 08    1410            STA (DPTRB),Y
032C- 88       1420            DEY
032D- A5 50    1430            LDA LINNUM
032F- D1 06    1440            CMP (DPTRA),Y
0331- 90 09    1450            BCC .1
0333- B1 06    1460            LDA (DPTRA),Y      A$ SHORTER THAN OR SAME
0335- 91 08    1470            STA (DPTRB),Y                           AS W
0337- A9 00    1480            LDA #0
0339- 91 06    1490            STA (DPTRA),Y
033B- 60       1500            RTS
               1510 *--------------------------------
033C- A8       1520 .1         TAY
033D- B1 FE    1530 .2         LDA (SPTRA),Y      LOOK AT SPLIT BOUNDARY
033F- C9 20    1540            CMP #$20           FOR A BLANK
0341- F0 07    1550            BEQ .3             FOUND ONE
0343- 88       1560            DEY
0344- D0 F7    1570            BNE .2             BACK UP THE SPLIT
               1580 *---NO BLANK IN W CHARS----------
0346- A5 50    1590            LDA LINNUM
0348- D0 04    1600            BNE .4         ...ALWAYS
               1610 *--------------------------------
034A- 98       1620 .3         TYA
034B- C8       1630            INY            SKIP OVER BLANK
034C- 84 50    1640            STY LINNUM
034E- A0 00    1650 .4         LDY #0         LENGTH OF B$
0350- 91 08    1660            STA (DPTRB),Y
0352- 38       1670            SEC
0353- B1 06    1680            LDA (DPTRA),Y      LENGTH OF A$
0355- E5 50    1690            SBC LINNUM
0357- 91 06    1700            STA (DPTRA),Y
0359- C8       1710            INY
035A- 18       1720            CLC
035B- B1 06    1730            LDA (DPTRA),Y
035D- 65 50    1740            ADC LINNUM
035F- 91 06    1750            STA (DPTRA),Y
0361- C8       1760            INY
0362- B1 06    1770            LDA (DPTRA),Y
0364- 69 00    1780            ADC #0
0366- 91 06    1790            STA (DPTRA),Y
0368- 60       1800            RTS
```

Enhancing Your Apple II (A Review)..................Bill Morgan

Don Lancaster, the well-known author of electronics books for the hobbyist (and a subscriber to AAL), has now entered the Apple arena in a big way. His latest book, "Enhancing Your Apple II, Vol. 1", promises to be the start of a long series of easy-to-use guides to the important internal workings of the Apple.

The main enhancements he offers in this volume are simple modifications to the Apple's video circuitry, to allow EXACT software access to the video timing. This permits your program to play all sorts of tricks with the display modes. There is also a wealth of information on the Apple's techniques of video storage and output.

The basic hardware modification is a single wire from an IC in the video circuitry to either the cassette or the game input. With this wire and a little bit of code, it is easy to switch display modes between screen scans, avoiding a lot of messy glitches on the screen. With more code, and careful timing, you can lock the processor to the display timing and switch between text and graphic modes (hi-res or lo-res) in mid-line.

There is also a very good 60-page chapter on disassembling and understanding other people's programs. Don presents a novel technique of color-coding a printout of a monitor disassembly, to bring out the structure of a program and the function of each routine. The example program is Apple's High-Res Character Generator, from the DOS Toolkit. He later uses the information discovered about the character generator and the Hi-Res display to develop a slower and smoother scrolling routine for Hi-Res text.

He shows us other enhancements, as well. There are two different ways to attach a modulator's output line to your TV set, avoiding that clumsy little switch box that the manufacture gives us. How about a programmable color-killer circuit? With this one you can have software control of color vs. black-and-white display. There are sections about generating extra colors, in both Hi-Res and Lo-Res graphics.

In the back of the book are postcards for sending feedback and ordering other materials. All the code in the book (26 programs) can be ordered on diskette, for $14.95. He uses the DOS Toolkit Assembler, but we plan to talk to him about providing the programs in S-C format. You can also order a kit of the parts for all of the hardware modifications he describes. That kit costs only $11.95 + shipping, from a dealer in Oklahoma. Future plans include more volumes of enhancements and a possible bulletin board system for updates to the books.

All in all, "Enhancing Your Apple II" looks to be an important and useful book. Like all of Lancaster's books, it is published by Howard W. Sams. It is 232 pages long, size 8 1/2 X 11 inches, and sells for $15.95. We have ordered a stock here at S-C, and will sell them for $15.00 + postage.

For Volume 2 of the "Enhancing" series, he has promised us more video techniques, a keyboard enhancer, something called an "Adventure Emergency Toolkit", graphics software for daisy-wheel printers, a two-dollar interface for the BSR controller, and much more. I'm looking forward to it!

This is a good time to mention another of Don's books, which has received too little attention. I am speaking of "The Incredible Secret Money Machine". Despite the title, it is not a get-rich-quick pamphlet, but rather a very, very useful guide to starting and operating a free-lance technical or craft business. "Money Machine" is 160 pages of tightly packed information on strategy and tactics, getting started, and dealing with customers, suppliers, and the government.

There is enough practical advice on communication, both verbal and graphic, to make up several courses in advertising and technical writing. Bob and I refer to this book regularly, and have long felt that it is one of the best books around for the budding entrepeneur. We have also ordered "Money Machine", and will sell it for $7.50 + postage.

Last minute addition: We just received a review copy of another new book from Don Lancaster, Micro Cookbook Vol. 1 - Fundamentals. This one is a very basic introduction to microcomputer principles. He talks about how to learn and what to learn, and introduces some hardware fundamentals. He also promises Vol. 2, about Machine-Language Programming. It looks very good; I'll have more details next month. We especially like this sentence at the end of the Preface: This book is dedicated to the 6502.

Last last-minute addition: Don sent me a copy of the program disk, and I am now converting the source files to S-C format. By the time you read this, he will have the S-C code. When you order the diskette from Synergetics (Lancaster's company), just mention that you want the version with the S-C files.

Quickie No. 4.............................Bob Sander-Cederlof

To print a two byte value in hexadecimal:

```
     LDA HI.BYTE
     LDX LO.BYTE
     JSR $F941
```

Clarification on Loading the RAM Card..........Paul Schlyter

Last month Bob S-C told how to use an EXEC command file without ENDing an Applesoft program. His example may have obfuscated the process of loading a file into a RAM card, because it really is not necessary to use an EXEC file for this purpose.

You can BLOAD into the RAM card without leaving Applesoft, contrary to Bob's information, by merely write-enabling it. The soft-switches $C081 and $C089 write-enable the RAM card (with bank 2 or bank 1 at $D000, respectively), leaving the motherboard ROMs read-enabled. This means everything you write goes into the RAM card, and everything you read comes from the motherboard ROMs. Thus you can simply BLOAD into the RAM card, and BLOAD will write to those addresses.

Here is a short program that loads the whole 16K RAM card, all from within a running Applesoft program, without EXEC files.

```
100 D$ = CHR$ (4)
110 B2 = 49281 : REM $C081 -- SELECT BANK TWO
120 B1 = 49289 : REM $C089 -- SELECT BANK ONE
130 P = PEEK(B2) + PEEK(B2) : REM WRITE ENABLE BANK TWO
140 PRINT D$"BLOAD LC.BANK 2"
150 P = PEEK(B1) + PEEK(B1) : REM WRITE ENABLE BANK ONE
160 PRINT D$"BLOAD LC.BANK 1"
```

[Note from Bob S-C: My face is red! Paul will note that I modified his program above in lines 130 and 150. He wrote "130 POKE B2, PEEK(B2)" and similarly for line 150. However, some RAM cards, such as my Andromeda board, will disable write-enable if the soft-switches are addressed during a write-cycle. The POKE does just that; therefore, I changed 130 and 150 to do two PEEKs in a row. Further, I recall when working with a Corvus drive last year that BLOADing from a Corvus into the RAM card did not work unless a monitor had already been copied into the space from $F800-$FFFF.]

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $13 postage for other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage).